Machine vision methods for cell manipulation

Elina Hiltunen

School of Electrical Engineering
27.9.2016

Supervisor: Prof. Quan Zhou
Instructor: Zoran Cenev
Machine vision methods can provide automatic solutions for cell manipulation. In this thesis implementations of machine vision algorithms were developed for recognizing and tracking objects in a video from a microscope setting. Those objects are a tip of the needle, cells and micro particles. For each object three different machine vision detection algorithms were tested within MATLAB®, the numerical computing environment. The performance of each implementation was evaluated and compared with each other. The results from the performed tests claim that the most suitable object detection method for the needle’s tip is the template matching and for the cells as well as the micro particles is the blob analysis.

Keywords: object recognition, tracking, needle, cell, particle, template matching, blob analysis
| **Tekijä:** Elina Hiltunen |
| **Työn nimi:** Machine vision methods for cell manipulation |
| **Päivämäärä:** 27.9.2016 | **Kieli:** englanti | **Sivumäärä:** 4+39 |

Sähkötekniikan ja automaation laitos

**Työn valvoja:** Prof. Quan Zhou  
**Työn ohjaaja:** Zoran Cenev


**Avainsanat:** object recognition, tracking, needle, cell, particle, template matching, blob analysis
Foreword

I want to thank my supervisor Prof. Quan Zhou who gave me this opportunity to finally complete my studies by offering a master’s student position and my instructor Zoran Cenev who instructed my work and encouraged me.

I also want to thank my family for support during my studies.

Espoo 27.9.2016

Elina Hiltunen
# Contents

Abstract................................................................................................................................. i  
Tiivistelmä ............................................................................................................................. ii  
Foreword................................................................................................................................ iii  
Contents................................................................................................................................. iv  

1 Introduction ......................................................................................................................... 1  

2 Background information ................................................................................................. 2  

3 Research material and methods ..................................................................................... 6  

4 Results ............................................................................................................................... 13  

4.1 The tip of the needle ...................................................................................................... 13  

4.1.1 Template matching ................................................................................................... 14  

4.1.2 Point tracker ........................................................................................................... 17  

4.1.3 Cascade object detector ......................................................................................... 20  

4.2 Cells ............................................................................................................................. 24  

4.2.1 Active contour ......................................................................................................... 24  

4.2.2 Hough circle transform ........................................................................................ 26  

4.2.3 Blob analysis ......................................................................................................... 29  

4.3 Particles ....................................................................................................................... 31  

4.3.1 Hough circle transform ........................................................................................ 31  

4.3.2 Blob analysis ......................................................................................................... 32  

4.3.3 Template matching ............................................................................................... 33  

4.4 Comparison and discussion ......................................................................................... 34  

5 Summary .......................................................................................................................... 37  

6 References ......................................................................................................................... 38  

7 Appendix ........................................................................................................................... 40
1 Introduction

Machine vision is a broad branch of technologies and methods that use information acquired from images to perform various tasks. Object recognition algorithms rely on matching, learning or pattern recognition algorithms that use feature-based techniques. In the selection of machine vision algorithms for this thesis the emphasis is on known algorithms that have a ready function in Matlab\textsuperscript{©}, a numerical computation environment. Matlab's computer vision system toolbox [1] provides a great variety of functions that can be used for machine vision applications. Background information about cell manipulation and object recognition and tracking is presented in the next chapter.

This thesis focuses on machine vision methods for recognizing and tracking of micro objects in cell manipulation. The thesis is related to the research done in Micro-and nanorobotics group in the Department of Electrical Engineering and Automation, in the School of Electrical Engineering at Aalto University. The purpose of the cell manipulation task in the focus is to insert single micro particles into individual living animal cells with a needle. The relevant objects in this task are one or more cells, the tip of the needle and one or more micro particles. It was expected that the recognition and tracking to be performed by machine vision algorithms. In this thesis, three methods are implemented for recognizing and tracking each of these relevant objects.

The main research question is to find machine vision algorithms that are useful for detecting and tracking objects in the specific cell manipulation task. The testing of the algorithms was done by implementing them in a Matlab. The findings of the thesis evaluate how implementations of the machine vision algorithms perform and their performances compared to each other. The research work was done using recorded experimental videos obtained from a micromanipulation setup developed around inverted microscope. The implemented algorithms were tested only with videos in Matlab, not with an actual microscope setting in the laboratory. The types of cells, particles and tip are limited to those used in the videos.
2 Background information

Cell manipulation

Cell manipulation is a branch of science where living cells are manipulated in various ways, such as positioning, grasping and injections. Cell manipulation combines knowledge of molecular biology and microtechnology, such as microrobotics and microsystems. Performing a cell manipulation task is difficult, so there has been research to make it easier and more successful with the knowledge from microtechnology and automation. Machine vision is a well-established and wide set of technologies and methods that provide ways to develop automated systems based on gathering data from images and videos. [2]

The process of manipulating single cells is widespread in the field of molecular biology. Most common applications are intracytoplasmic injections, pronuclei DNA injection, drug development, gene therapy and other biomedical areas. Cell manipulation methods also depend on whether the cells are adherent cells or suspended cells. Naturally cells grow either in adherent or suspended cell culture. Adherent cells are easier to manipulate and there are more commercial automated cell injection methods for adherent cells. Adherent cells grow attached to a surface, so they stay in place for the injection. Suspended cells grow flowing loosely in liquid medium, so injecting accurately, especially automatically, is a difficult task that can easily fail. The cell manipulation setting in this thesis is with adherent cells. [2][3]

The history of cell manipulation goes far back. Micromanipulation with pipette was developed in the early 1900s by Marshall Barber in the USA. His new technique was the microinjection, which could be used for cloning bacteria. The method begins with separating a single cell with a fine-pointed capillary pipette made of glass. The cell is touched with the tip of the pipette and by the capillary action the cell enters the pipette. The cell is released from the pipette by air pressure through the pipette. This capillary pipette method is still widely used for various applications, as it is simple and practical. The second part of Barber’s micromanipulation is a special pipette holder, an early micromanipulator, which allows three-dimensional movement of the pipette. With two pipettes, he could perform injections into cells. One pipette was used for holding the cell to be manipulated (with capillary action) and another used for injecting. This two pipette method is nowadays a standard method for cell injection into animal cells. [4]

There are various techniques for performing cell manipulation tasks. Existing cell manipulation techniques can be classified as contact manipulation, also referred to as mechanical micromanipulation, and non-contact manipulation. Mechanical micromanipulation techniques include injections which are usually done with pipette and magnetic micromanipulation. Conventionally, injections are performed manually by an experienced operator. After microelectromechanical (MEMS) technology has been developed, it has been offering important tools to manipulate cells. MEMS tools, such as microactuators and microrobots, can be used to achieve more precision and
efficiency. Non-contact manipulation techniques include laser trapping, electro-rotation and dielectrophoresis. [2][5]

Cell manipulation being a field of science with a long tradition, methods and techniques for it have been continually researched and developed. A traditional way of performing a cell manipulation task, for example cell injection, is difficult and requires experienced human operator, who performs the task only by looking at the image from the microscope. For a human operator, performing cell manipulation tasks can become straying for one’s eyes, which can lower the efficiency of performing cell manipulation tasks and lower the survival rate of the cells. Also, cells are usually fragile and they can be damaged during manipulation due to excessive force, hand tremor or contamination. [2][5][6]

To overcome these difficulties and to improve efficiency, various techniques and systems have been developed. Some of them combine tele-operated micromanipulators with guidance from machine vision, some of them are automated. Since everything happens under a microscope, machine vision is useful and practically the only good way to produce information and guidance for the cell manipulation task and achieve fully automated cell injection systems. Machine vision can be used for replacing the human operator to recognize correct injection locations and for providing accurate sensor input to control an automated injection pipette. [7][8]

One case example from the literature is presented below about how machine vision algorithms can be used for cell manipulation task. Ammi et al. [6] have developed a visual and haptic interface for cell injection. Their objective was to develop a virtual reality human-machine interface with vision- and force-based methods for realistic and augmented operator interaction, which would improve the process of manipulation of a biological cell. This narration focuses on the object recognition part of their paper. The manipulation task was an injection into a egg cell, which is illustrated in the figure 1. The egg cell was manually selected and held in place with a holding pipette and then injected with another pipette.

![Injection of the cell.](image)

The recognition of the cell and the pipette was considered as a template matching problem, and several approaches were introduced in the paper. For the cell’s
membrane recognition an active contour model was used. Because of the presence of different objects, such as the pipettes or impurities of the medium, images needed pre-processing before using the active contour, also called snakes, algorithm. Pipettes and impurities were removed from the image and after that a Sobel filter was applied to enhance the cell. The pipettes/membrane contact points were used to define the starting points for the snakes algorithm. Lastly, the recognized contours were labeled. To achieve good result, a threshold filter was applied beforehand to dispose light gradient variations. Once the cell was isolated in the enhanced image, the use of snakes algorithm for the outer membrane was easier and faster. The result of using snakes algorithms is illustrated in the figure 2.

![Figure 2 Results of the snakes algorithm for detecting cell's membrane. Initialization (left) and deformation (right).](image)

### Object recognition and tracking

Object recognition is a process of identifying a specific object in an image. The first phase of object recognition is to detect regions or points of interest from the image. That is done by detecting edges, regions and shapes. Then object recognition can be defined as a labeling problem based on models of known objects. An object recognition system typically has the following components, which are also presented in the figure 3: [9]

- **Model database** which contains all the models known to the system. Models contain information of features of the objects, such as size, color and shape.
- **Feature detector** which applies algorithms used to identify the features and their locations in the image.
- **Hypothesizer** which assigns likelihoods to objects present in the image. This step reduces the search space for the recognition by using certain features.
- **Hypothesis verifier** which uses object models to verify the hypotheses and refines the likelihood of the objects. The system then selects the object with the highest likelihood, based on all the evidence, as the correct object.
All the object recognition systems have the model database and use features for detection, but hypothesizer and verifier can be used or absent, depending on the complexity of the problem and the object recognition strategy. Some systems use only hypothesizer and then select the object straightforward with the highest likelihood as the correct object. Some systems use only verification, for example template matching. Different object recognition strategies are illustrated in the figure 4. [9]

Object detection and object recognition are terms used only for a single image. Tracking is defined as the process of detecting an object or multiple objects over time in a video stream. Tracked objects are usually also moving in the video stream. The detections of objects in the current frame are associated with the detections in previous frames. Tracking an object can be classified into two cases, depending on if the camera has a fixed pose or if the camera changes its pose. When the camera is in a fixed pose, the task of tracking an object is a problem of detecting and locating the object or objects in a stream of input images. [10][11]
3 Research material and methods

For developing and testing the implementations of machine vision algorithms several videos were used. The videos were recorded by Canon DSLR D 600 camera which was attached to an inverted microscope Zeiss® Axio A1. The camera is stationary in a fixed location. As operating with needle is delicate a task, usually all movement in the video is slow and there are only minor changes between successive frames. There is only limited set of objects in the video: a needle, cells and micro particles. Debris, scratches on the glass substrate and dust present in the fluidic environment are also present in the field of view, however, these objects represent noise in the system. Therefore, objects of interest are cells, tip of the needle and micro particles. The summary of videos used for developing and testing the implementations of machine vision algorithms is presented in the table 1. The more detailed description of the videos is told in the following text.

All of the videos with cells were older and of quite low quality compared to the videos of the needle and the particles. At first, three different videos of cells were used to develop and test implementations, but one of them was discarded as for being of too poor quality and therefore not representing the current quality of videos recorded from the microscope. As it was not possible to get new videos of cells, two other videos were used for testing, even though they were dated too.

There were good quality videos of the needle and it was also possible to record new videos. For testing the performance and errors new videos were recorded. In total, there are four videos with just the tip. In the first video the needle is moving horizontally from right to left and back and at each end point the needle is lowered 1 µm in perpendicular direction. The second video is similar, but at each end point the needle is lifted 1 µm in perpendicular direction. In the third and the fourth video the needle is not moving horizontally, but it is lowered or lifted 1 µm in perpendicular direction in every second. Lifting or lowering the needle will gradually make it out of focus, when the focus plane is the starting level and camera is kept focused on that level.

For particles there were at first the same kind of old videos as the cell videos, but they were discarded and new videos of new kind of particles were recorded. The particles are circle shaped and similar to each other. In some videos the particles are just staying in position and in some videos one particle is moving towards needle driven by magnetic force.

Programming the implementations was done within Matlab and in all implementations Matlab computer vision system toolbox was utilized. Machine vision algorithms used in this thesis are presented in the following.
Table 1 The summary of videos used for developing and testing the algorithm implementations

<table>
<thead>
<tr>
<th>The videos used for developing and testing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cell videos</strong></td>
</tr>
<tr>
<td><strong>Needle videos</strong></td>
</tr>
<tr>
<td>- needle moving horizontally and at each end point the needle is lowered 1 µm</td>
</tr>
<tr>
<td>- needle moving horizontally and at each end point the needle is lifted 1 µm</td>
</tr>
<tr>
<td>- needle is in a position horizontally and is lowered 1 µm in every second</td>
</tr>
<tr>
<td>- needle is in a position horizontally and is lifted 1 µm in every second</td>
</tr>
<tr>
<td><strong>Particle videos</strong></td>
</tr>
<tr>
<td>- video with 2 stationary particles</td>
</tr>
<tr>
<td>- video with 4 stationary particles</td>
</tr>
<tr>
<td>- video with 1 moving particle</td>
</tr>
</tbody>
</table>

**Hough circle transform**

Hough transform is a well-establish algorithm that can be used for feature extraction from an image. The purpose of the technique is to find instances of objects within a certain class of shapes, such as lines, circles or ellipses, by a voting mechanism. Hough circle transform is a specialization of the basic Hough transform. In the Hough transform’s voting mechanism each point on a curve votes for several combinations of parameters. The parameters that win a majority of the votes are declared the winners. [12]

The implicit equation for a circle is

\[(x - a)^2 + (y - b)^2 = r^2\]  \hspace{1cm} (1)

When the parametric equations for a circle are presented in polar coordinates the equations for the parameters a and b of the circle are:

\[a = x - r \cos \theta\]  \hspace{1cm} (2)

\[b = y - r \sin \theta\]  \hspace{1cm} (3)

The algorithm has four steps: [12][13]

1. Quantizing the parameter space for the parameters a and b

2. Accumulator array computation

   The parameter space is represented as an array of accumulators, representing discrete parameter values. At first, initializing of the all accumulators to zero is done. The magnitudes of gradient and angles \(\theta(x,y)\) are computed. The pixels
(points) with high gradient, which are edge pixels, are selected as being candidate pixels and start voting to the accumulator array. The classical voting pattern is full circle with fixed radius.

3. Center estimation

The votes of candidate pixels belonging to the actual circle accumulate in the accumulator array. Therefore the local maxima in the accumulator array correspond to centers of circles in the image. The center of the actual circle is at the intersection of voting patterns.

4. Radius estimation

If the same accumulation array is used for more than one radius value, radii of the detected circles have to be estimated separately. There are several algorithms for doing that.

The Matlab has a ready function for Hough circle transform (`imfindcircles`). Hough circle transform is used for detecting round cells and particles in the video frames.

**Template matching**

The basic idea of template matching is to detect instances, represented in a template image, in another image. The simple way of doing that is to place the template at a location in another image and detect its presence at that point by comparing intensity values of the template and corresponding values in the other image. Usually intensity values do not match exactly, so decision is done based on difference equations. The template is moved over the other image and for each location the intensity values are compared and their difference calculated. The locations that are local maxima and are above a certain threshold value, are the locations of the template. There are also more sophisticated methods to perform template matching that, for example, address the problem of variation caused by noise, different viewpoints or changes in illumination or imaging sensor [14], [12].

Matlab has a template matching system object TemplateMatcher which finds the best match of a template within an input image. The TemplateMatcher works by shifting a template over a region of interest or over the entire image. It has two search methods: Exhaustive and Three-step. Exhaustive method searches through whole image pixel by pixel. Hence, it is computationally heavy, but can produce more precise result. Three-step method is faster because it does not inspect every pixel as it uses a neighborhood approach. In that approach the step size is at the beginning larger, usually about equal to half of the maximum search range, and then step size is decreased as the search proceeds. In each step of the search nine points are being compared. At the each step of the search the best matching point of the nine is selected to be the center of the search for the next step. [15]

Template matching is primarily used for recognizing the needle tip, as it is of known shape. The shape although varies when it is not in the focus of the microscope.
Template matching is also used for one implementation for particles, but as the Matlab’s TemplateMatcher is used, that implementation can only detect one particle.

**Cascade object detector**

Cascade object detector utilizes algorithm developed by Viola and Jones [16]. The algorithm consists of three components which are an image representation as a summed area table [17], a weak learning algorithm and a cascade classifier. In their paper they used Haar like features, [18] which are simple rectangle shapes. Features are used for classifying images in the detection process. The features can be calculated rapidly from the image representation, which is called integral image. It is a representation in which each location is calculated as sum of pixels above and to the left of the location. Hence, the integral image can be computed in one step from the original image.

The learning algorithm is a variant of algorithm called AdaBoost [19] and it is used for selecting a small set of features and for the training the classifier. Boosting algorithms typically need also a weak learning algorithm which is then called repeatedly in the boosting process. Each time the weak learning algorithm is called with a different subset of the training samples, and then the result of that is a new weak prediction rule. Boosting process requires many rounds. The result of the boosting is a single prediction rule, which is supposed to be more accurate than any of the weak prediction rules it is combined from. In the Viola-Jones algorithm the weak learning algorithm selects a single rectangle feature which best separates the positive and the negative samples. To be able to classify several features, for each feature the optimal threshold classification function is determined individually by the weak learning algorithm. The purpose of the optimal classification function is that the minimum number of samples is misclassified. A weak classifier $h_j(x)$ by Viola and Jones consists of a feature $f_j$, a threshold $\theta_j$, and a polarity $p_j$ indicating the direction of the inequality sign: [16][19]

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

The third component of this algorithm is the use of boosted classifiers in cascading stages to create a fast detector which rejects the most of negative samples while detecting almost all positive instances. It is a degenerated decision tree where the training of a classifier happens in stages. At each stage classifier is trained to detect almost all instances of the determined feature while rejecting a certain amount of patterns that are not the determined feature. Simple classifiers in the first stages are used to reject the majority of the samples, as the assumption is that positive instances are rare, before the more complex following stages that are used to achieve low false positive rates. When a classifier returns positive result, the next classifier is triggered to do classification. A negative result at any stage leads to the immediate rejection of the sample. [16][17]
Matlab has a CascadeObjectDetector system object which uses pre-trained classifiers for detecting object categories from the image. It can be trained to detect object categories whose aspect ratio stays sufficiently invariable, such as faces and traffic signs. The training is done with a large amount of two kinds of images. The first set is called positive samples. They are images that have the object to be detected and those regions of interest need to be marked in the images. The second set is called negative samples and those are images that do not contain the object to be detected. In the training a cascade classifier uses several stages to learn to detect the object. All stages consist of a batch of weak learners, which are simple classifiers. Each stage is trained using boosting, which gives an accurate classifier as a result. To get accurate CascadeObjectDetector, parameters for training, which are the number of stages, the false positive rate, the true positive rate, and the type of features to use for training, need to be selected well. There are trade-offs between, for example number of stages and level of the false positive rate. When the training is complete, there is a classifier that is a xml-file that can be used for creating a CascadeObjectDetector. When used for detection, it performs multiscale object detection on the input image and returns a matrix of bounding boxes containing the detected objects. Cascade object detector is used for one implementation for the tip. [20]

**Active contour**

Active contour, also called snakes, is a method for finding contours and edges in images. The algorithm consists of deformable energy-minimizing spline, the snake, which is directed by external constraint forces, image forces and internal forces. The joint action of these forces causes the spline match the object contour in the image. The external constraint forces act to put the spline near the target location, the image forces push the spline towards salient features, like lines and edges and internal forces resist deformation of the spline. These forces need to be supplied for the snakes algorithm, they can come from user interface, additional automatic mechanism or higher-level interpretations. So the snakes algorithm only works if it gets beforehand information about the shape and approximate location of contour. Kass et al. [21] represent the position of a snake parametrically by \( \mathbf{v}(s) = (x(s), y(s)) \), its energy function can be written as

\[
E_{\text{snake}}^* = \int_0^1 E_{\text{snake}}(\mathbf{v}(s)) \, ds \\
= \int_0^1 E_{\text{int}}(\mathbf{v}(s)) + E_{\text{image}}(\mathbf{v}(s)) + E_{\text{con}}(\mathbf{v}(s)) \, ds
\]

where \( E_{\text{int}} \) represents internal energy of the spline due to deforming, \( E_{\text{image}} \) represents image forces and \( E_{\text{con}} \) represents external constraint forces. [21]

Matlab has a ready function ‘activecontour’ which segments a grayscale image into foreground and background using active contour model. The function returns a binary image where the foreground is white and the background is black. It also needs beforehand information about where the contours to segment are. That is done by
giving a binary mask as parameter for the function. The ‘activecontour’ function works in iteration so it is relatively slow. [22]

One problem to be solved in this thesis is how to set mask or starting region to find objects to be detected efficiently and accurately enough. Active contour algorithm is used for detecting cells, as they each form a contour. The active contour method turned out to be so slow that is was not used for detecting particles, as the implementations for them were made after the implementations for the cells.

Point tracking

Point tracking is basically tracking of features within an image stream. The tracking is based on finding the displacement of a feature from the previous image frame to the current image frame. Matlab’s Point tracker is a system object that tracks a set of points using the Kanade-Lucas-Tomasi (KLT), an iterative feature-tracking algorithm. KLT algorithm is based on two papers, first one, by Lucas and Kanade [23], resolves the translational image registration problem. They present a method that minimizes the sum of squared intensity differences between a previous and a current window. The idea is that as subsequent frames do not differ much from each other, the current window can be approximated by a translation of the previous frame. Also, the image intensities in the following frame can be written as those in the current frame added with a residue term that depends almost linearly on the translation vector. In the second paper, by Tomasi and Kanade [24], a feature tracking algorithm is developed. It is based on solving the image displacement. If the displacement is small enough compared to texture fluctuations within the region of interest, the displacement vector itself can be written approximately as the solution to a $2 \times 2$ linear system of equations. The basic step of the tracking procedure is to calculate that displacement.

Point tracker works particularly well for tracking objects that have clear features and whose shapes do not change. Matlab’s point tracker system object needs to be first initialized by specifying the initial locations of the points in the first video frame. The feature points can be found and selected with any method Matlab has for representing feature points. Calling for point tracker returns coordinates corresponding to the new locations of the points in the input frame. It also returns a logical array, which contains information whether or not each point has been reliably tracked.

Matlab’s point tracker implementation of the KLT algorithm uses image pyramids. The point tracker system object generates an image pyramid, where each level is reduced in resolution by a factor of two compared to the previous level. The reduction in resolution is done by down-sampling in every pyramid level by a factor of two the previous level in width and height. With these pyramid levels, algorithm can handle larger displacements of points between frames, although with larger number of levels the computational cost is also increased. The point tracker also has a forward-backward error threshold, the bidirectional error, which is calculated as the distance in pixels from the original location of the points to the final location after the backward tracking. When the maximum value for bidirectional error is set, points that cannot be
reliably tracked are eliminated. Point tracker method is used for detecting and tracking the needle’s tip. [25]

**Blob analysis**

Blob analysis is a method of detection of connected regions in an image. Blobs are regions that differ from the background of the image in brightness or in color. One of the first and also the most common version of blob analysis is based on the Laplacian of the Gaussian. The image is convolved with the Gaussian function and that gives the Gaussian scale-space representation of the image, to which is then applied Laplacian operator. It is also possible to compute the Laplacian of the Gaussian operator first and then convolved with the image to create the Laplacian of the Gaussian scale-space representation. A scale-space representation is a framework for dealing with image structures which naturally occur at different scales, introduced by Witkin and Koenderink [26]. It represents an image as one-parameter family of smoothed images, parametrized by the size of the smoothing kernel used for suppressing fine-scale structures. As the scale of standard deviation of Laplacian of the Gaussian increases, blobs converge to local extrema at some scale. This method can detect both dark and bright blobs. [27]

Matlab’s BlobAnalysis system object computes statistics for connected regions, blobs, in a binary image. It can be used for object detection as it computes and returns centroids and bounding boxes of the blobs. There are several parameters to be set to get blob analysis to work as wanted for different cases. As the blob analysis is done to a binary image, the video frames need to be thresholded into binary image first. Usually, some enhancement is also required to make regions clear. Blob analysis method is used for detecting particles and cells, as they both form blobs, closed areas that can be extracted from the image. [28]
4 Results

As a result of this thesis, there are three methods implemented for detecting and tracking the needle’s tip, cells and micro particles. Each implementation works primarily for the videos it was tested with. If the program code is to be used for different video it is possible that parameters need to be changed or other corrections made. This is because of the delicate nature of machine vision. In all implementations, the images are enhanced or converted into binary or grayscale images. If the other video is different, for example, has totally different lighting, it is possible that these enhancements and conversions do not work as intended. In all implementations the images are enhanced or converted into binary or grayscale images. If the other video is different, for example, has totally different lighting, it is possible that these enhancements and conversions do not work as intended.

In each section of this chapter each implementation is presented and its performance is evaluated and after that they are compared to each other. The program codes for each machine vision algorithm implementation are attached in the appendix.

4.1 The tip of the needle

For detecting and tracking the tip of the needle the following methods were used: template matching, point tracker and cascade object detector. The needle videos are of good quality and they were recorded for the purposes of this thesis. As the needle changes shape in the image as it goes out of the focus, it was tested how much the needle can be located above or below the focus plane while the detecting still works correctly. The change of focus is illustrated in the figure 5.

![Figure 5 The shape of the tip varies in different focus.](image-url)
4.1.1 Template matching

For template matching Matlab’s system object TemplateMatcher was used. It has two search methods, exhaustive and three-step. For this implementation the exhaustive search method is selected, because it finds the tip reliably. This heavy operation at each loop cycle for each frame of the video causes the method to be very slow. As a template, small image, cut from original video image, was used. In the template there is the tip of the needle and its size is 31 x 33 pixels. The size of the video image is 1088 x 1920 pixels, which is unnecessary large. To make processing of the video faster, the size of the images is reduced with Gaussian pyramid [29] to half of the original size. Gaussian pyramid uses lowpass filtering and down-sampling the image pixels to compute the reduction of an image. Figure 6 illustrates the block diagram of the execution of the template matching implementation. At first, the variables and system objects are initialized. The first frame is read to detect initial coordinates of the tip. Then the program enters the main loop in which TemplateMatcher is called. It returns the best match which is then marked into image, the resulted coordinates of the detection are saved and the image is played in the video player.

![Block diagram of the implemented algorithm for template matching](image)

Figure 6 The block diagram of the implemented algorithm for template matching

The testing the implementation was done with two kind of videos. In the first kind of videos, the needle moves horizontally right to left and back and at each end point the needle is lowered or lifted 1 µm in perpendicular direction. This causes it gradually become out-of-focus. In the second kind of video the needle is in a position and not moving horizontally, but it is lowered or lifted 1 µm in perpendicular direction in every second. In the figure 7 one example frame of tracking with template matching and the template image are presented.
When the tip of the needle is in focus, it is detected reliably but the execution of the program code is slow. When the tip of needle shifts out of focus, there becomes error in the detection of the tip. When the needle is lifted upwards from the focus plane, the image of the tip becomes softer and the detection is done downwards along the needle. When the needle is lowered downwards from the focus plane, the image of the tip becomes softer and there appear to be two tip ends. The detection is drawn to these false ends and detection also shifts between them. Error was measured using the second kind of videos, where the tip is moving only perpendicularly up or down relative to the focus plane. Normalized total error is illustrated in the figure 8. All errors are calculated relatively to the detection in the first frame, which is considered to be correct as the tip is in focus plane.
The tracking of the moving tip is reliable, when it is in sufficiently in-focus. From the total error it can be seen that detection is correct until the tip is lowered or lifted approximately 15 µm from the focus plane. First kind of videos where the tip moves from left to right and back were used to inspect the functioning of the tracking implemented with template matching. The average execution time of handling and displaying one frame (one while loop cycle) measured with Matlab’s tic-toc function is 0.5697 seconds, when every frame is used. When only every 10th frame was used, average execution time is 1.1937 seconds.
To improve the execution efficiency of this method the template matching search is limited to a smaller area than the whole image. The location of the tip in the video image is quite fixed and predictable. In this implementation, the search is limited to the region of interest that is a square of 100 pixels and which is located around the detection of the previous frame. This improvement makes execution of the code less heavy and faster and it works as reliably as when the search is done within the whole image. Average execution time of handling and displaying one frame is then 0.1218 seconds, when every frame is used. The video output is still running slower than just playing video without executing template matching, but compared to whole image search, it runs very smoothly. With more tuning, this template matching with small search area could be used for real-time applications, which have similar scenario of a slowly moving needle in fixed and predictable environment.

4.1.2 Point tracker

Matlab’s PointTracker systems object was used to implement the second method for tip recognition and tracking. PointTracker tracks a set of points using the Kanade-Lucas-Tomasi (KLT) feature-tracking algorithm. The implementation needs to be initialized with an object region from where to select feature points. It can be initialized in two alternative ways. Firstly, user can select the object region from the displayed image. Secondly, the program code uses template matching in the first frame of the video and the object region is selected around the detected point, which is the needle’s tip if the tip is in focus at the beginning of the video. The object region is selected to be 10 pixels larger than the template, 26 x 27 pixels, when the template in a reduced size is 16 x 17 pixels.

Feature points are selected with Matlab’s function detectMinEigenFeatures, which is one of the Matlab’s feature point detecting algorithms. It uses minimum eigenvalue algorithm developed by Shi and Tomasi [30] to find feature points. In this case, four to six feature points are found. As the object region is small and the tip is the only object there, program code calculates the mean of detected feature points and presents that as the center of the tip to be tracked. An example image of the point tracking algorithm in progress is represented in the figure 9.

Also, for this implementation the size of the video is reduced to half with Gaussian pyramid, similarly as for the template matching. As this method is to be compared with template matching, same videos, presented in the last chapter, are used.
The execution of the program code for the point tracker is presented in the figure 10. First, the initialization and setting up variables and systems objects is done. First frame of the video is read to get the object region from where to select feature points. There is an option in the code for getting object region and one of the options needs to be commented out. Then the program enters the main loop in which video is read frame by frame and the point tracker is called for every frame. Detection of the tip is calculated to be the center of all detected feature points. Detection is saved and markers drawn into image which is then played in the video player.

Figure 9 Tracking with the point tracker. Green marks are feature points and red mark is the calculated center.

Figure 10 The block diagram of the implemented algorithm for point tracker.
Performance of the point tracker method is faster than template matching that does the search to the whole image. When compared to template matching with small search area, the point tracker is almost as fast as it is. Average execution time of handling and displaying one frame (one while loop cycle) measured with Matlab’s tic-toc function is 0.1265 seconds, when every frame is used. When only every 10th frame was used, average execution time is 0.7968 seconds. Reliability and error of detection was measured similarly as with template matching. The normalized total error is illustrated in the figure 11. All errors are calculated relatively to the detection in the first frame, which is considered to be correct as the tip is in focus plane. From the total error it can be seen that detection is correct until the tip is lowered approximately 10 µm from the focus plane or lifted 6 µm. As the detection is a calculated center of all tracked feature points, it is not true detection of the tip and the detection of individual feature points can be more or less accurate.
4.1.3 Cascade object detector

As a third method for detecting and tracking the needle’s tip a cascade object detector was used. Matlab’s CascadeObjectDetector system object can be used for detecting and tracking objects in a video. At first, cascade classifier needed to be trained for detecting the tip. For training were used 260 positive samples, images that contain the tip. Those images were collected from the videos used for testing implementations.
The tip is the only object in those videos and its focus changes as the needle is gradually moved away from the focus plane. Of those selected 260 images some had the tip in-focus and some had the tip slightly off-focus, to train the classifier recognize the tip also when it is not perfectly in focus. The regions of interests were set manually with Matlab’s training image labeler. Regions were set to be small squares or rectangles around the tip as illustrated in the figure 12. Training algorithm used at most 254 positive samples per stage. Negative samples, images that do not contain the object to be detected, were collected from an “empty” video. There were no objects in the video, just some small impurities on the background. For the training there were 700 negative samples, but the training algorithm selected to use at most 508 negative samples per stage. Following parameters were selected for the training: 5 stages, false alarm rate of 0.25 and feature type ‘Haar-like’ features. Object training size was set automatically by the algorithm and it turned out to be good, close enough to actual size of the tip in the video. Parameters were set by experimentally. Training was fast and its output information is presented in the appendix 1. It stopped in the third stage and returned to the second stage. The training times per stage were: 25 seconds for 1\textsuperscript{st} stage, 22 seconds for 2\textsuperscript{nd} stage and 59 seconds for 3\textsuperscript{rd} stage. Very low false alarm rate $8.8767 \times 10^{-5}$ was reached in the third stage.

![Figure 12 An example of a region of interest for cascade object classifier training.](image)

In the actual implementation at first the cascade object detector is created with the xml-file that training a cascade classifier produced. It is also important to set parameters regarding the size of the object to be detected when creating the cascade object detector. The minimum size is set to 40 * 32 pixels and the maximum size is set to 50 * 40 pixels. When called, the cascade object detector returns bounding boxes for the objects detected. That is different from the template matching where returned information is coordinates of the center of the detection. The bounding box is a four-element vector that specifies in pixels, the upper-left corner and the size of a bounding box. The cascade object detector finds false detections in some frames. That is probably because the trained detector is not perfect. In the implementation, it is assumed that the correct detection of the tip is always in the first row of the bounding boxes matrix. With the video it was tested this seems to hold if the tip is enough in focus. When the tip is not in focus, cascade object detector founds also detection form two-forked glows that appear in the image. The error when the tip is moved out of the
focus plane was measured similarly to the case of template matching and point tracker implementations. The normalized total error is represented in the figure 13. The gaps in the error diagram are due to that some of the saved data needed to be discarded as it was from false detection. From the total error it can be seen that detection is correct, until the tip is lowered approximately 12 µm from the focus plane or lifted 15 µm.

![Graph 1](image1.png)

**Figure 13** Normalized total error of cascade object detector for the tip.
An example frame of the tracking of the tip with cascade object detector is shown in the figure 14. The average execution time of handling and displaying one frame (one while loop cycle) measured with Matlab’s tic-toc function is 0.1498 seconds, when every frame is used. When only every 10\textsuperscript{th} frame was used, average execution time is 0.8230 seconds. Execution of this implementation is rather slow.

Figure 14 Example of tracking the tip with the cascade object detector. Bounding box is drawn in yellow.

The block diagram of the execution of the program code is illustrated in figure 15. The training for the cascade object detector is done separately and the result of that is a xml-file that is used to initialize the CascadeObjectDetector system object. First, initializing of the variables and system objects is done. Then the program enters the main loop in which the cascade object detector is called. It returns the bounding box for detected object which is then drawn into image and also saved. Lastly, the image is played in the video player.
4.2 Cells

For detecting and tracking cells following methods were used: active contour, Hough circle transform and blob analysis. For testing implementations there were no very good videos of cells available. Developing and testing was done mainly with two videos that are quite different. The first one has two round cells and a needle, and its main color is red and there is a certain amount of noise in the video. That video was used for developing and testing the active contour method and the Hough circle transform method. The second video has three big and round cells of which two are just partly in the picture. The main color of the video is white or beige and the cells are somewhat off focus. This video was used for testing blob analysis method for cells. The created implementations are tuned for these videos, so they might need some tuning of parameters or other corrections if used with totally different videos.

4.2.1 Active contour

Matlab’s ‘activecontour’ function was used for this implementation of active contour method for detecting cells. To work properly, it needs information about where the contour to segment is beforehand. That information is passed to the function via giving a mask as parameter. ‘Activecontour’ function segments a grayscale image into foreground and background. The output image is a binary image where the foreground is white and the background is black.

The implementation works frame by frame. Image frame in the processing is first converted into a grayscale image and then sharpened with unsharp masking. For sharpening Matlab’s ‘imsharpen’ function with parameters ‘Amount’ = 4, ‘Radius’ = 0.5 was used. These values of parameters give a strong sharpening effect but with small area around the edge pixels.
The implementation can detect one, two or three cells, and a user must define beforehand how many cells there are to be detected. The user also needs to select cells from the first image frame of the video with precise rectangles. These selections are used as masks for the 'activecontour' function. Other parameters used for the function are: maximum number of iterations to perform in evolution of the segmentation = 100, active contour method used for segmentation = 'Chan-Vese', 'ContractionBias' = 0.3 and 'SmoothFactor' = 1.3. Chan-Vese method is the Chan and Vese's region-based energy model, which can detect objects which’s boundaries are not necessarily defined by gradient [31]. The contraction bias defines the tendency of the contour to grow outwards or shrink inwards. Positive values bias the contour to shrink inwards. The selected contraction bias causes the segmentation to be clearly inside or on the edge of the cell, not outside of the cell. The smooth factor defines the degree of smoothness or regularity of the boundaries of the segmented regions. Smoothing is required in this case so that the edge of the segmentation is as circle shaped as possible. Each cell is detected first separately from the image and then all segmented binary images are combined into one binary image. From this combined binary image the edges of the detected cells are extracted and those edge pixels are changed to yellow in the original image frame, which is then displayed in the video player. The flow diagram of the program code is represented in the figure 16. At first, initializing of the variables and systems objects is done. The selection of the number of cells to be detected needs to be done beforehand. The program asks for user to set the starting regions. Then program enters the main loop in which ‘activecontour’ function is called. The segmented binary images it returns are combined into one and the detections are marked into image and also saved. Then image is played in the video player.

This implementation finds selected cells fine if the selection with a rectangle is done precisely. However, the found edges are not as regularly circle shaped as cells actually are. There can be uneven edges or little bit of outside of the cell may be included or some sections of cells are left out. An example frame is illustrated in the figure 17. The main problem with this implementation is that it is very slow. The average execution time of handling and displaying one frame (one while loop cycle) with 2 cells,
measured with Matlab’s tic-toc function, is 4.6209 seconds, when every frame is used. When only every 10th frame was used, average execution time is 4.9141 seconds. The slowness is due to iteration nature of the ‘activecontour’ function and that it is executed up to three times for each frame. It also is not fully automatic as it needs beforehand information about the location of the region to be segmented and that is achieved by asking the user for that information.

![Cell detection with the active contour method](image)

**Figure 17** Example of cell detection with the active contour method.

### 4.2.2 Hough circle transform

Hough circle transform was used for detecting circle shaped cells. For this implementation Matlab’s ‘imfindcircles’ function was used. This implementation works only with videos that have a clear background, have cells to be detected that are circle shaped and have not got much noise. The video is processed frame by frame and each frame is first converted into a grayscale image and then into a binary image with Matlab’s system object Autothresholder with parameter 'ThresholdScaleFactor' = 0.80. Result of this conversion is that cells and other objects show up as black areas and the background is white. This is illustrated in the figure 18. For each frame ‘imfindcircles’ function is called with parameters 'ObjectPolarity' = 'dark' and 'sensitivity' = 0.86. Radius range of circles to be found is set to from 20 pixels to 40 pixels. All the cells in this video fit in that range. As the binary image is that way that the objects are black and the background is white, object polarity parameter is selected to be dark instead of bright. Sensitivity parameter defines the sensitivity for the Hough circle transform accumulator array. It is a factor between 0 and 1 and larger number increases the detection of circles, including weak and partially obscured circles. Higher sensitivity value also increases the risk of false detection, so the selection of sensitivity factor is a trade-off between how weak circles it can find and the accuracy. For this implementation the sensitivity factor is set to as high as possible to detect cells which
are not perfect circles in the binary image and still keeping accuracy that false detections are not made.

![Image](image.png)

Figure 18 Binary image after autothresholding. `imfindcircles` function is called for this kind of image.

Detected circles, which are the cells, are marked in the original image frame, which is then displayed in the video player. As the detected circles are presented with the center and the radius of the circle, drawn circles in the video are perfect circles, not uneven as in the active contour method. An example frame is illustrated in the figure 19. One problem with this Hough circle transform method is that on each frame the found circle can be slightly different from the previous frame which causes the drawn circle to blinkingly change in the video. Another known problem with this implementation is that in some frames all the cells are not detected which causes the cell’s drawn circle to disappear for a frame in the video. The execution of the code is very fast compared to active contour method, but slower compared to blob analysis method. The average execution time of handling and displaying one frame (one while loop cycle) measured with Matlab’s tic-toc function is 0.3769 seconds.
The block diagram of the program code is presented in the figure 20. The first variables and system objects are initialized. In the main loop, the video is read frame by frame and for each image the ‘imfindcircles’ function is called. It returns the centers and radii of the found circles, which are then drawn into image and saved. Lastly, the image is played in the video player.

Figure 19 Example of detecting cells with Hough circle transform.

Figure 20 The block diagram of the implemented algorithm for Hough circle transform.
4.2.3 Blob analysis

At first, the blob analysis method was developed for the particles, but it also works for cells, if size parameters are changed. For implementation Matlab’s blob analysis systems object was used. When creating a blob analysis system object, the parameters need to be set properly. As this implementation uses only centroid information, other options for return value, which are blob areas and bounding boxes, can be set to false. So the only return value is centroid. Most important in this implementation is to set minimum and maximum blob areas as correct as possible. This prevents false detections. Minimum and maximum values for blob area were found out experimentally, but can also be measured from the image. Good values for cells in the white colored video are: 'MinimumBlobArea', 9000 and 'MaximumBlobArea', 200000. There is too much noise in the red colored video, so this implementation does not work for it. The maximum count of blobs can be set to a small number, if only some of the cells are to be detected. Mainly, it is ok to have a larger number for this parameter, so all cells are detected.

In the program code, the frames are read from the video file one by one. Each frame is first enhanced with morphological dilation and image arithmetic operations to make edges of the cells clearer. Then image frame is autothresholded into a binary image and all holes are filled. The blob analysis system object is then called for this filled binary image. An example of filled binary image is shown in the figure 21. Blob analysis returns centroids of the detected blobs, in this case, cells. Centroids are then saved into Matlab’s cell array and marked with red marks in the output video. An example frame is shown in the figure 22. Execution of this implementation is relatively fast. It is faster than two other methods for cell detection. Average execution time of handling and displaying one frame (one while loop cycle) measured with Matlab’s tic-toc function is 0.1378 seconds. Reliability of this method depends on how well the conversion into binary image success. If the cells stand out as complete white areas, blobs, they are reliably detected. Problem with blob analysis with this video it was tested with, was that even though the cells are staying in one position, the shape of the cell in the thresholded binary image is slightly different in each frame. That causes the detection of the centroid to be changing around the real center. Also, as the detected centroid is the center of the blob, it is not the real center of the cells that are only partially visible in the video. Different videos will probably need slightly different enhancements.
Figure 21 Filled binary image for blob analysis. Blob analysis is called for this kind of image.

Figure 22 Blob analysis for cells. Centroids of the cells marked with red mark.

The block diagram of the program code is represented in the figure 23. At first, initializing of variables and system object is done. In the main loop the video is read frame by frame. Each image frame is enhanced by morphological dilatation and image arithmetic operations before thresholding into a binary image. System object BlobAnalysis is called that binary image and it returns the centers of the detected blobs. Those centers are marked into the image and also saved. Lastly, the image is played in the video player.
4.3 Particles

For detecting and tracking particles were used following methods: Hough circle transform, blob analysis and template matching. The particle videos are of good quality and there were many videos to choose from. For developing and testing, three different videos were used: one with four particles, one with two particles and one with a particle that moves.

4.3.1 Hough circle transform

As the particles studied in this thesis, are also round in shape, Hough circle transformation method works well for them, too. Similar restrictions apply also to the particle videos as apply to the cell videos: the background in the video needs to be clear and there should not be too much noise. More detailed reporting about the implementation is in the part concerning the cell detection. For detecting particles, the radius range parameter needs to be changed. The good radius range for particles is from 10 pixels to 20 pixels. Particles show in the thresholded binary image as black rings on the white background, so the parameter 'ObjectPolarity' = 'dark' is correct for particles also. Hough circle transform method for particles is reliable if conversion into a binary image is successful, that is, particles stand out as complete black circles and there are no other impurities that resemble circles. The blinking and no detections in some frames are not problem with particle videos, as they are of better quality. There is no noise and the background is clearer, so the binary image is more stable. Examples of particle detection with Hough circle transform are represented in figure 24.

The average execution time of handling and displaying one frame (one while loop cycle) measured with Matlab’s tic-toc function is 0.7438 seconds. That is much slower than the execution time of same code with the video with which it was tested to detect cells. This is probably because of the size of the video. The video of round cells is older and its resolution is smaller than videos of particles. The resolution of particle videos is 1088 x 1920 pixels and resolution of cell video is 720 x 1280 pixels.
4.3.2 Blob analysis

The same blob analysis implementation works also for detecting particles, as works for detecting cells. The parameters for minimum and maximum blob area need to be changed to match the size of the particles in the video. Actually, these parameters need to be tested or calculated beforehand for all different videos, as the tighter they are, the better the results will be. Tight area range eliminates the detection of too small impurities in the video while avoiding the detection of too large objects such as a needle. For the two videos used for testing, particles’ areas are about 500-1500 and 700-5000. These values are selected from experiments. More detailed reporting about the implementation is in the part concerning cell detection. An example frame from the blob analysis for particles is illustrated the figure 25. This blob analysis implementation is also capable of tracking moving particles. This was tested with a video in which a particle is drawn to the needle by magnetic forces. The moving particle is detected and therefore tracked until it moves upwards and out of the focus plane. Average execution time of handling and displaying one frame (one while loop cycle) measured with Matlab’s tic-toc function is 0.2766 seconds. That is faster than Hough circle transform method. Overall, blob analysis seems to be best method for detecting and tracking particles, as it can detect many particles, and is relatively fast and reliable.
For the template matching implementation of particle the same Matlab’s system object TemplateMatcher as for the detection of the tip was used. Its disadvantage is that it can only detect one best match for the template. The program code for the tip works for the particle also when the template image is changed. When testing with a video that has two similar particles that are staying in place, the template matcher founds one of them and then the detection stays with that particle, because of the small search area. The search area is set to be a square of 100 pixels around the first detection. When testing with a video that has one particle that moves and then disappears from the image, the template matcher first tracks it normally. When it disappears, the template matcher tries to detect it and returns its best match from the
small search area, which obviously is not a correct match. Examples of detecting particle with template matching are shown in the figure 26. Average execution time of handling and displaying one frame (one while loop cycle) measured with Matlab’s tic-toc function is 0.1254 seconds. That is relatively fast, as fast as detecting the tip.

![Figure 26 Template matching for particle. Another particle is detected (left). A particle has been disappeared and there is then a false detection (right).](image)

4.4 Comparison and discussion

All implementations can detect and track the desired object or objects, at least for the videos they were tested with. Some of them work better than the others and some work less efficiently than others. For the tip was done the most measuring of error, caused by un-focused image. For cells and particles, accuracy and reliability were approximated by looking at the performance of the program code. For all the implementations, the execution time of single loop cycle was measured, as the time for handling one frame. That measurement was done with Matlab’s tic and toc functions which are a simply stopwatch for measuring elapsed time. The average execution times were calculated from three test runs of the code. This comparison is also represented in the table 2.

The best method for detecting and tracking the needle’s tip is template matching with small search area, square that is set around previous detection. Template matching is the most accurate method of these three and it is also fast due to that small search area improvement. The point tracker is not perfect for detecting the tip, as it only tracks feature points, does not return one set of coordinates for the tip. In this implementation that is bypassed so that the center coordinates are calculated to be the average of those feature points. In this case that works, as the feature points are located around the edges of the tip. Cascade object detector returns a bounding
box that surrounds the detected object. So that also is not exactly the coordinates of
the tip, though the center of the bounding box can be calculated. False detections are
also a bit of a problem with cascade object detector, as only one detection is saved per
frame, so there might come false detections into that data.

For the cells, it was difficult to get good results, as the videos were of low quality.
The best method for detecting cells is blob analysis, as it was the fastest of the three.
The accuracy seemed to be quite similar between Hough circle transform method and
blob analysis method. And they were also tested with different videos, as neither of
the videos work for both methods. Active contour method was very slow compared to
every other method and its accuracy was not good either.

The best method for detecting and tracking particles is blob analysis. It can
reliably detect all the particles in the image. It is not the fastest implementation of the
three, but a disadvantage with the fastest, template matching, is that it can only detect
one particle. Hough circle transform method was slow for the particle video. That is
probably because of the size of the video. If video frames are reduced in size, it can
become faster.

From all of these machine vision algorithm implementations, I recommend the
template matching and blob analysis for using in cell manipulation tasks in scenarios
similar to ones in these videos.
### Table 2 Comparison of implementations.

<table>
<thead>
<tr>
<th></th>
<th>accuracy and reliability</th>
<th>computational speed in seconds (execution time of handling one frame)</th>
<th>automatic or user input needed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tip</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>template matching</td>
<td>detection is correct when less than 15 µm away from the focus plane</td>
<td>0.1218</td>
<td>automatic</td>
</tr>
<tr>
<td>point tracker</td>
<td>detection is correct when less than 10 µm below or 6 µm above the focus plane</td>
<td>0.1265</td>
<td>automatic, but can use user input</td>
</tr>
<tr>
<td>cascade object detector</td>
<td>detection is correct when less than 12 µm below or 15 µm above the focus plane</td>
<td>0.1498</td>
<td>automatic</td>
</tr>
<tr>
<td><strong>Cells</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>active contour</td>
<td>depends on the quality of user input and does not found edges of the cell smoothly</td>
<td>4.6209</td>
<td>needs user input</td>
</tr>
<tr>
<td>Hough circle transform</td>
<td>depends on the quality of the video and does not found same circle in every frame</td>
<td>0.3769</td>
<td>automatic</td>
</tr>
<tr>
<td>blob analysis</td>
<td>finds all cells but the coordinates of the center vary between frames</td>
<td>0.1378</td>
<td>automatic</td>
</tr>
<tr>
<td><strong>Particles</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hough circle transform</td>
<td>Better accuracy than with the cell video. Founds edges correctly and detection does not vary much between frames</td>
<td>0.7438</td>
<td>automatic</td>
</tr>
<tr>
<td>blob analysis</td>
<td>Better accuracy than with the cell video. Stable detection, does not vary much between frames</td>
<td>0.2766</td>
<td>automatic</td>
</tr>
<tr>
<td>template matching</td>
<td>Founds only one particle. If particle is not present in the image, produces false detection</td>
<td>0.1254</td>
<td>automatic</td>
</tr>
</tbody>
</table>
5 Summary

The goal of this thesis was to implement machine vision algorithms for cell manipulation for recognizing and tracking objects in a video from a microscope. The setting of cell manipulation task this thesis relates to, includes images of cells, a needle and micro particles. The purpose was to development three machine vision implementations for tracking of each type of objects: the tip of a needle, the cells and the particles. The main research question was to find and select machine vision algorithms and implement them into a Matlab program. Another important part of this thesis was also to evaluate the performance of each implementation. Selected machine vision algorithms are Hough circle transform, template matching, cascade object detector, active contour, point tracker and blob analysis.

For detecting and tracking the tip of the needle template matching, point tracker and cascade object detector were used. All of the implementations work, but the template matching was better than the two other methods. For template matching Matlab’s TemplateMatcher was used and to improve the execution time of the code, the search area was set to be a small square around the previous detection. For point tracker and cascade object detector Matlab’s systems objects from computer vision toolbox were also used.

For detecting the cells active contour, Hough circle transform and blob analysis were used. All of them were implemented using Matlab’s systems objects from computer vision toolbox. One problem with developing and testing was that there were no good videos of cells, where two old videos were used. All implementations worked, although especially active contour was very slow and none of them were very accurate. The approximated level of accuracy was similar for both Hough circle transform and blob analysis methods. As the blob analysis was fastest of the three, it was selected to be the best.

For detecting and tracking the particles Hough circle transformation, blob analysis and template matching were used. All of them were implemented using Matlab’s systems objects from computer vision toolbox. The implementations using Hough circle transform and blob analysis can detect multiply particles, whereas template matching can detect only one limited by Matlab’s TemplateMatcher. All of the implementations work. The best implementation is blob analysis, as the detection is stable and it was faster than Hough circle transform method.

From all of these machine vision algorithm implementations, I recommend the template matching and blob analysis for applications in cell manipulation tasks in scenarios similar to the videos tested.
6 References


7 Appendix

Appendix 1  Training log of the cascade object detector

Appendix 2  Template matching program code

Appendix 3  Point tracker program code

Appendix 4  Cascade object detector program code

Appendix 5  Hough circle transform program code

Appendix 6  Active contour program code

Appendix 7  Blob analysis program code

Appendix 1 Training of the cascade object classifier

Automatically setting ObjectTrainingSize to [33, 32]
Using at most 254 of 260 positive samples per stage
Using at most 508 negative samples per stage

Training stage 1 of 5
[-----------------------------------------------]
Used 254 positive and 508 negative samples
Time to train stage 1: 25 seconds

Training stage 2 of 5
[-----------------------------------------------]
Used 254 positive and 508 negative samples
Time to train stage 2: 22 seconds

Training stage 3 of 5
[-----------------------------------------------]
Very low false alarm rate 8.87867c-05 reached in stage.
Training will halt and return cascade detector with 2 stages
Time to train stage 3: 95 seconds

Training complete
Appendix 2 Template matching program code

%% Template matching for pipette’s tip
VideoSize = [720 1280];
filename = 'MVI_9524.MOV'; %Write the name of the video file here
tipvideo1 = vision.VideoFileReader(filename);
videoPlayer = vision.VideoPlayer;
videoPlayer.Position([4 3]) = VideoSize;
videoPlayer.Position([1 2]) = [50 200];
gaussPyramid = vision.Pyramid('PyramidLevel', 1);
%template matcher and template processing.
matcher=vision.TemplateMatcher('SearchMethod', 'Exhaustive');
T = imread('particle_template.tif');
T = rgb2gray(T);
T = im2single(T);
T = step(gaussPyramid, T);
framenumber = 0;
t = zeros(1,1000);
yCoordForPlot = zeros(1,10000);
xCoordForPlot = zeros(1,10000);
indexForPlot = 1;
% This part is used for the first frame case and for recording coordinates of detection
first = step(tipvideo1);
firstbw = rgb2gray(first);
first_reduced = step(gaussPyramid, firstbw);
initialLoc=step(matcher,first_reduced,T);
%Selecting size of the region of interest and calculating the left-up corner for roi for
%the first case
x = 100; %width and height of the roi
a = round(x/2);
uppercorner = [initialLoc(1,2)-a initialLoc(1,1)-a];
while ~isDone(tipvideo1)
    tic
    imageframe=step(tipvideo1);
    imageframebw = rgb2gray(imageframe);
    imageframebw_reduced = step(gaussPyramid, imageframebw);
    %the first frame case
    if(framenumber == 0)
        Loc_small=step(matcher,imageframebw_reduced(uppercorner(1,1):(uppercorner(1,1)+x),uppercorner(1,2):(uppercorner(1,2)+x)),T);
        Loc = [(uppercorner(1,2)+Loc_small(1,1)) (uppercorner(1,1)+Loc_small(1,2))];
    %otherwise this is used
    else
        Loc_small=step(matcher,imageframebw_reduced(Loc(1,2)-a:(Loc(1,2)+a), Loc(1,1)-a:(Loc(1,1)+a)),T);
        Loc = [((Loc(1,1)-a)+Loc_small(1,1)) ((Loc(1,2)-a)+Loc_small(1,2))];
    end
    markedimage = insertMarker(imageframebw_reduced, Loc, '+', 'Color', 'red');
yCoordForPlot(1,indexForPlot) = Loc(1,2);
xCoordForPlot(1,indexForPlot) = Loc(1,1);
indexForPlot = indexForPlot + 1;
step(videoPlayer, markedimage);
t(indexForPlot) = toc;
framenumber = framenumber+1;
end
release(videoPlayer);
release(tipvideo1);
Appendix 3 Point tracker program code

```matlab
%% PointTracker for pipette's tip
VideoSize = [720 1280];
filename = 'MVI_9354.MOV'; %Write the name of the video file here
tipvideo1 = vision.VideoFileReader(filename);
videoPlayer = vision.VideoPlayer;
videoPlayer.Position([1 2]) = VideoSize;
videoPlayer.Position([1 2]) = [50 200];
gaussPyramid = vision.Pyramid('PyramidLevel', 1);
tracker = vision.PointTracker('MaxBidirectionalError', 1);

%template matcher and template processing
matcher = vision.TemplateMatcher;
T = imread('tip-template2-small.tif');
T = rgb2gray(T);
T = im2single(T); %template matcher and template processing
T = step(gaussPyramid, T);
[h, w] = size(T);
t = zeros(1,100);

%creating vectors for coordinates and an index
yCoordForPlot = zeros(1,10000);
xCoordForPlot = zeros(1,10000);
indexForPlot = 1;

%% Reading first frame for initialization
frame = step(tipvideo1);
framebw = rgb2gray(frame);
framebw_reduced = step(gaussPyramid, framebw);
%objectRegion from template matching
Loc_template = step(matcher, framebw_reduced, T);
objectRegion = [Loc_template(1,1)-(h/2)-5 Loc_template(1,2)-(w/2)-5 w+10 h+10];

%Selecting feature points
points = detectMinEigenFeatures(framebw_reduced, 'ROI', objectRegion);
initialize(tracker, points.Location, framebw_reduced);
while ~isDone(tipvideo1)
    tic
    frame = step(tipvideo1);
    framebw = rgb2gray(frame);
    frame_reduced = step(gaussPyramid, framebw);
    %Calculation for center of all valid points which is used as the detection
    index = 1;
count = 0;
sumx = 0;
sumy = 0;
s = size(validity,1);
    for index= 1:s
        n = validity(index,1);
        if n == 1
            sumx = sumx + points(index,1);
            sumy = sumy + points(index,2);
            count = count + 1;
        end
    end
    loc = [round(sumx/count), round(sumy/count)];
    region = [loc(1,1)-15 loc(1,2)-15 30 30];
    %case for drawing the initial detection. See below.
    if indexForPlot == 1
        firstLoc = loc;
    end
    %recording coordinates
    yCoordForPlot(indexForPlot) = loc(1,2);
xCoordForPlot(indexForPlot) = loc(1,1);
    indexForPlot = indexForPlot + 1;
    out = insertMarker(frame_reduced, loc, '+', 'Color','red');
    out = insertShape(out, 'Rectangle', region,'Color', 'red');
    out = insertMarker(out, points(validity,:), '+');
    step(videoPlayer, out);
t(indexForPlot)=toc;
end
release(videoPlayer);
release(tipvideo1);
```
Appendix 4 Cascade object detector program code

%% Cascade object detector for pipette's tip
VideoSize = [720 1280];
filename = 'MVI_9354.MOV'; %Write the name of the video file here
-tipvideo1 = vision.VideoFileReader(filename);
 videoPlayer = vision.VideoPlayer;
 videoPlayer.Position([4 3]) = VideoSize;
 videoPlayer.Position([1 2]) = [50 200];
 %Creating CascadeObjectDetector with pre-trained xml-file and defining the size of the
detectable object
 TipDetector = vision.CascadeObjectDetector('tipDetectorHaar.xml', 'MaxSize', [50 40], 'MinSize', [40 32]);
t = zeros(1,1000);
boundingBoxes = zeros(1000,4);
index=0;
while ~isDone(tipvdeo1)
tic
 imageframe=step(tipvideo1);
 %Call for CascadeObjectDetector
 bbox=step(TipDetector,imageframe);
detectedImg = insertShape(imageframe, 'rectangle',bbox);
index=index+1;
 %Recording bounding boxes. Assuming that correct detection is always in the first
row
 boundingBoxes(index,:) = bbox(1,:);
 step(videoPlayer, detectedImg);
t(index)=toc;
end
release(videoPlayer);
release(tipvideo1);

Appendix 5 Hough circle transform program code

%% Tracking of cells or particles with Hough circle transform
VideoSize = [720 1280]; %Write the name of the video file here
video1 = vision.VideoFileReader(filename);
 videoPlayer = vision.VideoPlayer;
 videoPlayer.Position([4 3]) = VideoSize;
 videoPlayer.Position([1 2]) = [50 200];
 % Select radius range. For cells [20 40] and for particles [10 20]
 radiusRange = [10 20];
circles = zeros(100,3);
 autothres = vision.Autothresholder('ThresholdScaleFactor', 0.80);
 index = 1;
savedCircles = cell(1000,1);
t = zeros(1,1000);
i = 1;
while ~isDone(video1)
tic
 frame = step(video1);
 framebw = rgb2gray(frame);
 framebin = step(autothres, framebw);
 [centers, radii] = imfindcircles(framebin, radiusRange, 'ObjectPolarity', 'dark',
 'sensitivity', 0.86);
 if size(centers) > 0
 circles = [centers(:,1) centers(:,2) radii(:)];
 markedImage = insertShape(frame, 'circle', circles, 'LineWidth', 1);
 savedCircles{index,1} = circles;
end

 index = index+1;
 step(videoPlayer, markedImage);
t(i) = toc;
i = i+1;
end
release(videoPlayer);
release(video1);
Appendix 6 Active contour program code

%% Tracking cells with active contour algorithm
VideoSize = [720 1280];
filename = 'MVI_7165.MOV'; %Write the name of the video file here
cellvideo = vision.VideoCapture(filename);
videoPlayer = vision.VideoPlayer;
videoPlayer.Position([4 3]) = VideoSize;
videoPlayer.Position([1 2]) = [50 200];
%Reading first frame for selecting starting contours
imageframe = step(cellvideo); %NOTICE! change to wanted number of cells to be detected.
% Can only be 1, 2 or 3.
numberOfCells = 2;
str = sprintf('Select %d cells by precise rectangles', numberOfCells);
figure
title(str);
hold on
imshow(imageframe);
startRegion = zeros(numberOfCells, 4);
for n = 1:numberOfCells
    startRegion(n, :) = round(getPosition(imrect));
end
mask = false(size(rgb2gray(imageframe)));
for i = 1:numberOfCells
genvarname('segmented', num2str(i));
end
t = zeros(1, 100);
centroids = cell(1000, 1);
index = 1;
while ~isDone(cellvideo)
tic
imageframe = step(cellvideo);
imageframebw = im2single(rgb2gray(imageframe));
Isharp = imsharpen(imageframebw, 'Amount', 4, 'Radius', 0.5);
%Performing active contour for each cell separately and then combining segmented images together
for n = 1:numberOfCells
    mask(startRegion(n,2):startRegion(n,2)+startRegion(n,4), startRegion(n,1):startRegion(n,1)+startRegion(n,3)) = true;
    segmented = activecontour(Isharp, mask, 100, 'Chan-Vese', 'ContractionBias', 0.3, 'SmoothFactor', 1.3);
eval([segmented num2str(n) = segmented;]);
end
%Combining segmented images together
if numberOfCells == 1
    sum = segmented1;
elseif numberOfCells == 2
    sum = segmented1 | segmented2;
else
    sum = segmented1 | segmented2 | segmented3;
end
s = regionprops(sum, 'Centroid');
centroids(index, 1) = centers;
perim = bwperim(sum, 8);
red = imageframe(:,:,1);
green = imageframe(:,:,2);
blue = imageframe(:,:,3);
red(perim) = 255;
green(perim) = 0;
blue(perim) = 0;
out = cat(3, red, green, blue);
out = insertMarker(out, centers, '+', 'Color', 'yellow');
step(videoPlayer, out);
t(index) = toc;
index = index + 1;
end
release(videoPlayer);
release(cellvideo);
Appendix 7 Blob analysis program code

```matlab
%% Blobanalysis for particles or for cells
VideoSize = [720 1280];
filename = 'MVI_7326.MOV'; % Write the name of the video file here
video1 = vision.VideoFileReader(filename);
videoPlayer = vision.VideoPlayer;
videoPlayer.Position([4 3]) = VideoSize;
videoPlayer.Position([1 2]) = [50 200];
avtothres = vision.Autothresholder('ThresholdScaleFactor', 0.8);
% Creating a BlobAnalysis system object.
% Define the minimum and maximum blob area
blobanalyzer = vision.BlobAnalysis(...
    'AreaOutputPort', false, ...
    'BoundingBoxOutputPort', false, ...
    'OutputDataType', 'single', ...
    'MinimumBlobArea', 9000, ...
    'MaximumBlobArea', 2000000, ...
    'MaximumCount', 10);
objects = cell(1000,1);
i = 1;
t = zeros(1,1000);
while ~isDone(video1)
    tic
    frame = step(video1);
    framebw = rgb2gray(frame);
    % combination of morphological dilation and image arithmetic operations to enhance
    % the image
    y1 = 2*framebw - imdilate(framebw, strel('square',7));
    y1(y1<0) = 0;
    y2 = imdilate(y1, strel('square',7)) - y1;
    framebin = step(autothres, y2);
    framebinFilled = imfill(framebin, 'holes');
    % call for BlobAnalysis
    Centroids = step(blobanalyzer, framebinFilled);
    markedimage = insertMarker(frame, Centroids, '+', 'Color', 'red');
    step(videoPlayer, markedimage);
    objects{i,1} = Centroids;
t(i) = toc;
i = i + 1;
end
release(videoPlayer);
release(video1);
```